BIRCH, STEWART, KOLASCH & BIRCH, LLP

TERRELL C BIRCH
RAYMOND C STEWART
JOSEPH A KOLASCH
JAMES M SLATTERY
BERNARD L SWEENEY*
MICHAEL K MUTTER
CHARLES GORENSTEIN
GERALD M MURPHY, JR.
LEONARD R SVENSSON
TERRY L. CLARK
ANDREW D MEIKLE
MARC S. WEINER
JOE McKINNEY MUNCY
ROBERT J KENNEY
DONALD J DALEY
JOHN W. BAILEY
JOHN A CASTELLANO, III
GARY D YACURA

OF COUNSEL:
HERBERT M BIRCH (1905-1996)
ELLIOT A GOLDBERG*
WILLIAM L GATES*
EDWARD H. VALANCE
RUPERT J BRADY (RET.)*

*ADMITTED TO A BAR OTHER THAN VA

INTELLECTUAL PROPERTY LAW
8110 GATEHOUSE ROAD
SUITE 500 EAST
FALLS CHURCH, VA 22042-1210
U S A
(703) 205-8000

FAX (703) 205-8050
(703) 698-8590 (G IV)

e-mail. mailroom@bskb.com
web: http://www.bskb.com

CALIFORNIA OFFICE.
COSTA MESA, CALIFORNIA

THOMAS S. AUCHTERLONIE
MICHAEL R CAMMARATA
JAMES T. ELLER, JR
SCOTT L. LOWE
MARK J NUELL, PH.D.
D. RICHARD ANDERSON
PAUL C LEWIS
W KARL RENNER
MARK W MILSTEAD*
JOHN CAMPA*

REG PATENT AGENTS.
FREDERICK R HANDREN
MARYANNE ARMSTRONG, PH D
MAKI HATSUMI
MIKE S RYU
CRAIG A. McROBBIE
GARTH M. DAHLEN, PH.D.
LAURA C LUTZ
ROBERT E. GOOZNER, PH D
HYUNG N SOHN
MATTHEW J LATTIG
ALAN PEDERSEN-GILES
JUSTIN D KARJALA
C KEITH MONTGOMERY

Date: March 17, 2000

Docket No.: 0630-1060P

BOX PATENT APPLICATION
Assistant Commissioner for Patents
Washington, DC 20231

Sir:

Transmitted herewith for filing is the patent application of

Inventor(s): Min-Seok JANG

For: METHOD FOR IMPLEMENTING EVENT TRANSFER SYSTEM OF REAL TIME OPERATING SYSTEM

Enclosed are:

- ☒ A specification consisting of EIGHTEEN (18) pages
- ☒ FOUR (4) sheet(s) formal drawings
- ☒ An assignment of the invention
- ☒ Certified copy of Priority Document(s)
- ☒ Executed Declaration (☒ Original ☐ Photocopy)
- ☐ A statement (☐ original ☐ photocopy) to establish small entity status under 37 C.F.R. § 1.9 and 37 C.F.R. § 1.27
- ☒ Preliminary Amendment
- ☐ Information Disclosure Statement, PTO-1449 and reference(s)
- ☐ Other:

The filing fee has been calculated as shown below:

			LARGE ENTITY	SMALL ENTITY
	BASIC FEE		$690.00	$345.00
	NUMBER FILED	NUMBER EXTRA	RATE FEE	RATE FEE
TOTAL CLAIMS	16- 20 =	0	X 18 = $0.00	x 9 = $0.00
INDEPENDENT CLAIMS	1- 3 =	0	x 78 = $0.00	x 39 = $0.00
☐ MULTIPLE DEPENDENT CLAIMS PRESENTED			+ $260.00	+ $130.00
		TOTAL	$690.00	$0.00

☒ A check in the amount of $730.00 to cover the filing fee and recording fee (if applicable) is enclosed.

☐ Please charge Deposit Account No. 02-2448 in the amount of $0.00. A triplicate copy of this transmittal form is enclosed.

☒ Please send correspondence to:

BIRCH, STEWART, KOLASCH & BIRCH, LLP **or** Customer No. 2292
P.O. Box 747
Falls Church, VA 22040-0747
Telephone: (703) 205-8000

If necessary, the Commissioner is hereby authorized in this, concurrent, and future replies, to charge payment or credit any overpayment to Deposit Account No. 02-2448 for any additional fees required under 37 C.F.R. § 1.16 or under 37 C.F.R. § 1.17; particularly, extension of time fees.

Respectfully submitted,

BIRCH, STEWART, KOLASCH & BIRCH, LLP

By _____
Joseph A. Kolasch, #22,463
P.O. Box 747 Reg No. 35,416
Falls Church, VA 22040-0747
(703) 205-8000

JAK:jcp
0630-1060P
Attachments

(Rev. 01/08/2000)

PATENT
0630-1060P

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANTS: Min-Seok JANG

NEW APPLICATION GROUP:

FILED: March 17, 2000 EXAMINER:

FOR: METHOD FOR IMPLEMENTING EVENT TRANSFER SYSTEM OF REAL TIME OPERATING SYSTEM

PRELIMINARY AMENDMENT

Assistant Commissioner for Patents
Washington, D.C. 20231

March 17, 2000

Dear Sir:

Prior to examination of the subject application, please amend as follows:

IN THE CLAIMS:

Please amend the claims as follows:

Claim 4, line 1, delete "or 3".

Claim 5, line 1, delete "or 3".

Claim 6, line 1, delete "or 3".

Claim 9, line 1, delete "or 8".

Claim 10, line 1, delete "or 8".

Please add the following new claims:

--12. The method of claim 3, wherein as a result of the check whether the event value exists, when the event value exists, the event value is obtained from the event control block buffer, and the task routine is executed by the sort of the event.

13. The method of claim 3, wherein as a result of the check, when the event value does not exist, the current task is blocked and queued into the waiting-list of the event.

14. The method of claim 3, wherein when the kernel system function of receiving the event starts, a step for checking a validity of the event ID is further included for thereby generating an error code in the case of invalidity, and the routine is returned from the kernel system function.

15. The method of claim 8, wherein as a result of the check that whether the task exists in the waiting-list, when the waiting task does not exist, an event value is stored in the event buffer of the event control block.

16. The method of claim 8, wherein as a result of the check whether the task exists in the waiting-list, when the waiting task exists, an event value is transferred to the head task of the waiting-list.--

REMARKS

Entry of the above amendments is earnestly solicited.

Claims 1-16 are pending.

An early and favorable action on the merits is earnestly solicited.

In the event that any outstanding matters remain in this application, Applicant requests that the Examiner contact Gary D. Yacura (Reg. No. 35,416) at (703) 205-8071 to discuss such matters.

If necessary, the Commissioner is hereby authorized in this, concurrent, and future replies to charge payment or credit any overpayment to Deposit Account No. 02-2448 for any additional fees required under 37 C.F.R. §§ 1.16 or 1.17; particularly, extension of time fees.

Very truly yours,

BIRCH, STEWART, KOLASCH & BIRCH, LLP

By _____
Joseph A. Kolasch
Reg. No. 22,463

for No. 35,416

JAK:jcp

P.O. Box 747
Falls Church, VA 22040-0747
(703) 205-8000

METHOD FOR IMPLEMENTING EVENT TRANSFER SYSTEM OF REAL TIME OPERATING SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a real time operating system kernel, and in particular to a method for implementing an event transfer system of a real time operating system kernel.

2. Description of the Background Art

Most operating systems are capable of supporting a multi-tasking feature that a plurality of tasks are concurrently executed in accordance with a user's program for an efficient use of a system. A kernel is the core program of the operating system and is run generally in the privileged mode of the system, invoked by interrupts or system calls. In addition, the above-described kernel is composed of the essential and primitive functions of an operating system such as generation and scheduling of the tasks, communication and synchronization between tasks, basic memory management, interrupt services, and device driver interfaces, etc. The characteristics and performance of the operating system are basically dependent on the kernel.

The real time operating system is adapted to a system in which certain tasks need to be executed in real time. The kernel should assure the real time task to be carried out in a certain dead-line. Therefore, the real time operating system kernel uses a preemptive scheduling mechanism generally based on a priority

order.

The communication and synchronization between the tasks are important functions for the scheduling method and its performance and are implemented as a transfer system of events such as a message transfer between tasks using the queue, and the semaphore management for the mutual exclusion and synchronization between tasks. This event transfer system must be designed and implemented based on a minimization and optimization of a reaction time and on a priority-based preemptive scheduling.

In addition, since most of systems which require real time characteristics are embedded systems it is needed to reduce unnecessary functionalities and architecture of the systems and to implement the core functions that guarantee simple and effective operations.

As a solution for satisfying the above-described requirements, there can be the methods using a waiting-list to define and implement the relationship between an event and the tasks trying to obtain it.

The known event transfer system implementation method of an operating system kernel which uses a waiting-list will be explained with reference to the accompanying drawings.

Figure 1 is a view illustrating an example of the state of the tasks in a waiting-list of an event in the case that the known event transfer system implementation method for an operating system kernel is adapted.

As shown in Figure 1, the first task (task 1) has a role of repeating event sending (transfer) to the event control block ECB1 of a message queue. The priority values of the first task, and second through fourth tasks 2-4 that repeat event reception are 40, 30, 20 and 20, respectively, (the lower value has the

higher priority), and the second through fourth tasks 2-4 try to receive event value from a buffer of the event control block ECB1 of the first task.

When the first task (sender task), the event control block 1 of the first task, and the second through fourth tasks 2-4 are sequentially created and started to execute at a certain time interval, first of all, the first task is created and started to execute, and then the first task creates the event control block 1.

The event control block is a real structure of an event and is formed of a data structure managed by the kernel. Creating an event control block means that a task creates an event of its own. In the case that the second task 2 with the priority higher than that of the first task is created and started to execute, when the second task 2 calls the kernel system function of waiting for (receiving) an event, since the first task transfers no event yet, the second task 2 is blocked to the wait state waiting for the event to be transferred (sent) and is queued into the waiting-list of the event control block 1.

By the same reason, even though the third and fourth tasks 3 and 4 also have the higher priorities, the third and fourth tasks 3 and 4 are blocked respectively to the wait state waiting for the event and are queued into the waiting-list.

Figure 1 illustrates the waiting-list state at the above-described moment.

Thereafter, when the first task starts to send the event, at first the second task receives to obtain the event, is woke up and is resumed execution. Namely, the event is first transferred to the second task 2 first queued to the waiting-list based on the FIFO(First-In-First-Out), so that the second task is executed. Therefore, it causes the third and fourth tasks 3 and 4 having relatively higher priorities not to be first performed. Namely, the above-described method may be

adapted to a known round robin scheduling method, but it may not satisfy the scheduling mechanism which supports a real time characteristics.

In order to overcome the above-described problem, the structure of the waiting-list should be formed in a doubly linked list, and when an event is sent, the task with the highest priority should be searched in the waiting-list and resumed to execute. However, in this case, since it's not until an event is sent that the searching the waiting-list is performed, it may take more reaction-time. And the more number of the tasks waiting for the event, the more time it takes also.

SUMMARY OF THE INVENTION

Accordingly, it is an object of the present invention to provide a method for implementing a simple and efficient event transfer system using the waiting-list of a real time operating system kernel in which a task with higher priority first obtains an event and is executed.

To achieve the above object, a method for implementing a simple and efficient event transfer system according to the present invention is characterized in that a plurality of tasks trying to receive a certain event call a kernel system function for obtaining the event under the multi-tasking environment in which the priority-based preemptive scheduling is adapted, the real time operating system kernel beforehand queues the tasks into the waiting-list of the event in the order of the priorities. After this state, when the event occurs, the task having the highest priority in the waiting-list obtains the event and is woke up to resume execution.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will become better understood with reference to the accompanying drawings which are given only by way of illustration and thus are not limitative of the present invention, wherein:

Figure 1 is a view illustrating an example of a waiting-list state of tasks with respect to an event in which a known event transfer system implementation method of an operating system kernel is adapted;

Figure 2 is a view illustrating an example of a waiting-list of tasks with respect to an event in the case that an event transfer system implementation method of a real time operating system kernel is adapted according to the present invention;

Figures 3A and 3B are views illustrating the internal procedures of the kernel system function-calls as an example of an event transfer system implementation method of a real time operating system kernel according to an embodiment of the present invention; and

Figure 4 is a view illustrating creation, waiting, wake-up and execution state of each task of Figure 2.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The method for implementing the event transfer system of a real time operating system kernel according to the present invention will be explained with reference to the accompanying drawings.

Figure 2 is a view illustrating an example of a waiting-list state of tasks

with respect to an event in the case that the event transfer system implementation method of a real time operating system kernel is adapted according to the present invention, Figures 3A and 3B are views illustrating the internal procedures of the kernel system function-calls of the event transfer system implementation method of a real time operating system kernel according to an embodiment of the present invention, and Figure 4 is a view illustrating a creation, waiting, wake-up and execution state of each task of Figure 2.

As shown in Figure 2, it is defined that the priority of the first task is 40, the priority of a second task 102 is 30, the priorities of third and fourth tasks 103 and 104 are 20, respectively, (the lower values have the higher priority), and that the second through fourth tasks 102 through 104 are trying to receive the event which the first task transfers periodically through an event control block 101 of the first task.

As shown in Figure 4, after the first task creates the event control block 101 at the point A, if the second through fourth tasks 102 through 104 are created and started to execute at a certain interval at the points B, C, and D, the second through fourth tasks 102 through 104 call a kernel system function of waiting for an event from the event control block 101 of the first task. In this state, since the first task did not transfer any event, the second through fourth tasks 102 through 104 are blocked respectively, and become a wait state and are queued into waiting-list of the event control block 101 of the first task at the points B', C' and D'.

The task order queue in the waiting-list at the time when the second through fourth tasks 102 through 104 have been all blocked (E in the Figure 4), is not the sequence of calling the kernel system function as shown in Figure 1. Instead, the order of the third task 103, the fourth task 104 and the second task

102 which is the priority order of tasks as shown in figure 2. When a kernel system function call is performed for receiving an event by each task but none of event is sent yet, each task is blocked and is set to wait state. At this point, the priority of each task is checked and the task is inserted into the position of priority order of the waiting-list so that the waiting-list becomes the priority order. Therefore, the waiting-list is always maintained in the state of the priority order that the task with the highest priority is placed at the head of the waiting-list.

When the first task calls a kernel system function of transferring (sending) the event from the point F of Figure 4, the priority check is not additionally needed with respect to the tasks of the waiting-list, and the task with the highest priority at the head of the waiting-list is picked up from the list, and the event value is transferred to the task, so that the task is resumed execution. It means that since the waiting-list is already aligned in the higher priority order, when transferring the event, the task with the highest priority of the waiting-list first obtains (receives) the event by merely waking up and resuming the head-positioned task.

As shown in Figure 4, the event transferred by the first task is alternately obtained by the third task 103 and the fourth task 104 having the highest priority equally, and the second task 102 which has relatively lower priority does not obtain the event.

The internal procedures of the kernel system functions according to the present invention will be explained with reference to Figures 3A and 3B.

The method for implementing the event transfer system of a real time operating system kernel is implemented as kernel system functions and is formed of a pair of a kernel system function of receiving (waiting for) the event and a kernel system function of transferring (sending) the event.

As shown in the above embodiment, if each task calls the kernel system function of receiving the event or calls the kernel system function of sending the event, each task performs as the above-described operations in accordance with the internal procedures of the system functions.

First, the kernel system function of receiving (waiting for) in order for the task to obtain an event will be explained.

When a certain task (the current task) calls the kernel system function of receiving the event, the scheduling is temporarily disabled so that a context switch does not occur to other tasks in Step S2, and it is checked whether the argument ID of the event that the task receives is valid in Step S3.

As a result of the check, if the event ID is invalid, it means an error situation that the event-receiving attempt is performed with respect to the non-existing event. Therefore, the scheduling is enabled in Step S12, and the current task is returned from the kernel system function with an error code in Step S13.

As a result of the check, if the event ID is valid, next it is checked that whether the event value has been already transferred (sent) in Step S4. If the value exists, the event value is obtained from the buffer in the event control block in Step S5, and the process routine is performed by the kind of the event in Step S11, and the scheduling is enabled in Step S12, and the current task is returned from the kernel system function with the event value in Step S13.

Namely, as a result of the event value check, the fact that the event value exists means that the other task had already transferred the event. Therefore, the current task which has called the kernel system function of receiving the event does not need to be queued into the waiting-list to wait for the transmission of the event. Instead, the current task can directly get the event from the event buffer of

the event control block.

As a result of the event value check, if the event value does not exist, the current task which called the kernel system function of receiving (waiting for) the event is adjusted to the wait state in Step S6 and is queued into the waiting-list of the event in Step S7.

Here, the process for queuing the current task which called the kernel system function into the waiting-list in Step S7 will be explained.

/* if the waiting-list is empty (any task that waits for a corresponding event does not exist), or if the priority of the current task is lower than the priority of the tail portion task of the waiting-list or is the same(the priority of the current task is lower than any other tasks already queued with the standby list), it is directly inserted into the rear end (tail) of the waiting-list. */

```
if the head task of waiting-list = NULL
        then shortCut = TRUE
        else if priority of current task >= priority of tail task of waiting-list
                then shortCut = TRUE
                else shortCut = FALSE
if shortCut = TRUE
        then currentTask is inserted into the rear end (tail) of waiting-list
                return SUCCESS;
```

/* in the case that more than one task exist in the waiting-list, the priorities are compared, and the current task is inserted so that the tasks of the waiting-list

are arranged in the priority order. */

```
prevTask = NULL;
nextTask = head task of waiting-list;
while (nextTask is not NULL)
{
        if priority of currentTask >= priority of nextTask
                then prevTask = nextTask
                        nextTask = the next task of the nextTask in the
                        waiting-list
                else insert currentTask between prevTask and nextTask
                        return SUCCESS;
}
return FAIL;
```

The detailed procedure of Step S7 is devised by optimizing the process that the queuing is performed for thereby maintaining the priority order of the waiting-list when the task is inserted into the waiting-list of the event. Thereafter, the event is directly transferred to the task (the head (leading) task of the waiting-list) having the highest priority without the additional priority search when the kernel system function of sending the event is called.

Next in Step S8, it is checked whether a time-out condition is provided when the current task calls the kernel system function of receiving the event. If the time-out is designated, the system timer-related function of the kernel is set in order that the task should stop waiting for the event and wake-up itself in the case

of receiving no event by timeout due.

When the stopped scheduling is re-enabled in Step S9, the preemption may occur.

The current task (which has called the kernel system function) which is currently being executed by now is changed to the wait state and is queued into the waiting-list of the event as above described, and is removed from the scheduling candidates of the kernel. Therefore, now when the scheduling is resumed, the kernel executes another task which satisfies the execution condition such as the highest priority among the tasks of the ready state queued in a ready-list of the kernel in Step S10. Namely, the context switch occurs between the current task which is blocked in the wait state and another task which satisfies the execution condition. Namely, a series process of preemption in which the resource used by the current task is occupied by the other task and the execution control is also switched to the other task occur in Step S10.

The situation that n-number of tasks is queued in the waiting-list of a certain event at a certain point means that the n-number of the tasks all have passed through the steps S1 through S4 and steps S6 through S10. In the above-described embodiment, the second through fourth tasks 102 through 104 have passed through the steps S1 through S4 and steps S6 through S10 and become the state of Figure 2 (point E of Figure 4).

The head (leading) task of the waiting-list, namely, the task having the highest priority among the tasks of the wait state is woke up and is resumed execution when another task which is supported to send the event calls a kernel system function of sending the event or the time-out is elapsed to thereby cause the preemption, and the task routine is returned from the kernel system function of

receiving the event based on the steps S11 through S13.

While, the internal details of the kernel system function of sending (transferring, posting) the event will be explained.

When a certain task which is currently being executed calls the kernel system function of sending the event in Step ST1, the scheduling is temporarily disabled like the kernel system function of obtaining the event in Step ST2, and it is checked whether the argument ID of the event that the current task sends to is valid in Step ST3. As a result of the check, if the event ID is invalid, it means that an error situation that the event-sending attempt is performed with respect to the non-existing event. Therefore, the scheduling is re-enabled in Step ST12, and the current task routine is returned from the kernel system function with the error code in Step ST13.

As a result of the check, if the event ID is valid, next it is checked whether the waiting task exists in the waiting-list of the event in Step ST4. If any task of the wait state does not exist with respect to the event, the event value is simply stored in the event buffer of the event control block in Step ST5, and the event kind-based process routine may be additionally performed in Step ST11, and then the scheduling is re-enabled in Step ST12, and the current task routine is returned from the kernel system function in Step ST13.

In the above-mentioned additional event storing process, in the case that there is the duplication of event value or that the exceeding value needs to be prevented by checking the state of the event buffer of the event control block based on the several sort of event, or that the overflow of the event buffer needs to be prevented, the actual storing process may be avoided and thereby generate an error code.

As a result of the task existence check of the waiting-list, if the wait state task exists, the event value is directly transferred (stored) to the event buffer of the head (leading) task of the waiting-list (namely, the task having the highest priority in the waiting list) in Step ST6, and the head (leading) task is removed from the waiting-list in Step ST7. If the time-out condition had been set, it is released in Step ST8 and then the head (leading) task is changed to the ready state in Step ST9, and queued into the ready-list (queue) of the kernel in Step ST10.

At this time, since the tasks already have been queued in the waiting-list in the priority order when the kernel system function of receiving (waiting for) the event before in Step S7, it is not necessary to search the task having the highest priority in the waiting-list.

Thereafter, if the scheduling is re-enabled in Step ST12, the preemption can be occurred, so that the task (which was the head (leading) task in the waiting-list), which has been set ready to wake up in the step ST10 is resumed execution.

In the method for implementing the event transfer system of a real time operating system kernel according to the present invention, the task with the highest priority first can receive (obtain) the event and is executed, so the event transfer system which is suitable for the priority-based preemptive scheduling is provided to user's real time application programs.

In addition, the method for implementing the event transfer system of the real time operating system kernel according to the present invention is directed to the pre-process waiting-list management system in which the priority of the task is checked and the task is queued into the waiting-list in the priority order when the task waits for the event. And in the present invention, the reaction time is

decreased in the case that the event occurs, and the performance is enhanced for thereby avoiding the excessively complicated elements. Therefore, the present invention is easily adapted to the real time operating system kernel for the embedded systems.

As the present invention may be embodied in several forms without departing from the spirit or essential characteristics thereof, it should also be understood that the above-described embodiment is not limited by any of the details of the foregoing description, unless otherwise specified, but rather should be construed broadly within its spirit and scope as defined in the appended claims, and therefore all changes and modifications that fall within the meets and bounds of the claims, or equivalences of such meets and bounds are therefore intended to be embraced by the appended claims.

What is claimed is:

1. In a method for implementing an event transfer system of a real time operating system kernel under a multi-tasking environment in which a priority-based preemptive scheduling is adapted, a method for implementing an event transfer system of a real time operating system kernel which is characterized in that in the case that a plurality of tasks call a kernel system function of receiving an event with respect to one event under the multi-tasking environment, the tasks are blocked and previously inserted into a waiting-list of the event in a higher priority order, and in the case that the event transfer occurs, the task having the highest priority in the waiting-list obtains the event, is woke up and is resumed execution.

2. The method of claim 1, wherein said waiting-list of the event is managed based on the higher priority order so that the task having the highest priority is arranged at the most leading portion (head) of the waiting-list.

3. The method of claim 1, wherein when the kernel system function of receiving the event starts, it is checked whether there is an event value already sent.

4. The method of claim 1 or 3, wherein as a result of the check whether the event value exists, when the event value exists, the event value is obtained from the event control block buffer, and the task routine is executed by the sort of the event.

5. The method of claim 1 or 3, wherein as a result of the check, when the event value does not exist, the current task is blocked and queued into the waiting-list of the event.

6. The method of claim 1 or 3, wherein when the kernel system function of receiving the event starts, a step for checking a validity of the event ID is further included for thereby generating an error code in the case of invalidity, and the routine is returned from the kernel system function.

7. The method of claim 5, wherein when the current task is queued into the waiting-list, a time out option is additionally set if it exists.

8. The method of claim 1, wherein when transferring the event, it is checked whether the waiting task exists in the waiting-list of the event.

9. The method of claim 1 or 8, wherein as a result of the check that whether the task exists in the waiting-list, when the waiting task does not exist, an event value is stored in the event buffer of the event control block.

10. The method of claim 1 or 8, wherein as a result of the check whether the task exists in the waiting-list, when the waiting task exists, an event value is transferred to the head task of the waiting-list.

11. The method of claim 10, wherein said head task in the waiting-list

which receives the event value is adjusted to the ready state and is inserted into the ready list, and the additional process routine by the sort of the event is executed.

5

ABSTRACT OF THE DISCLOSURE

In the method for implementing the event transfer system of a real time operating system kernel, the task with the highest priority first obtains the event under the multi-tasking environment which requires a real time characteristics. In the case of the multi-tasking environment in which the priority-based preemptive scheduling is adapted, if a plurality of tasks with respect to one event, call a kernel system function of receiving the event, the real time operating system kernel queues the tasks into the waiting-list of the event in the priority order. In this state, when the event is sent, the task having the highest priority in the waiting-list immediately obtains the event, is woke up and is resumed execution.

FIG. 1
CONVENTIONAL ART

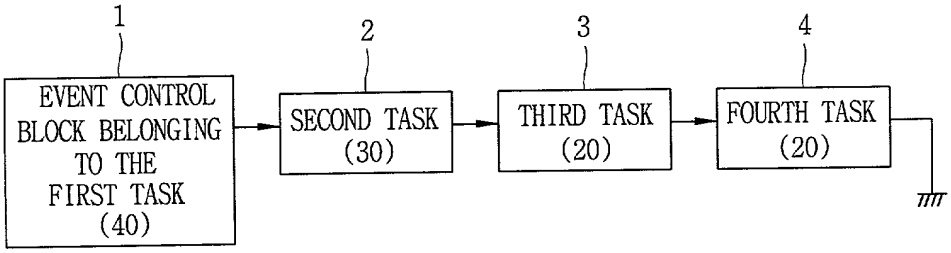


FIG. 2

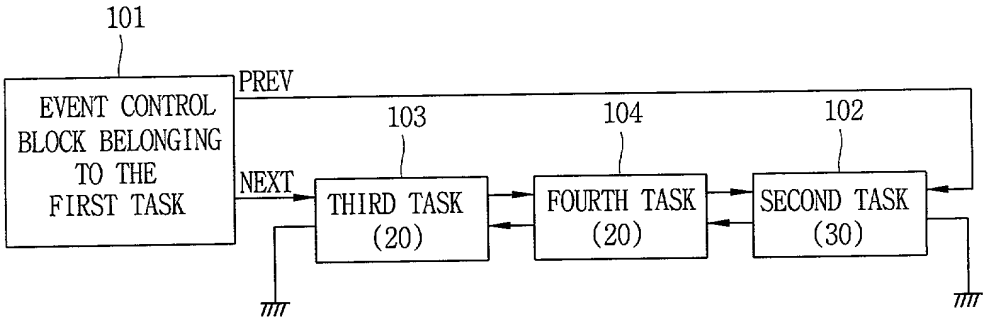


FIG. 3A

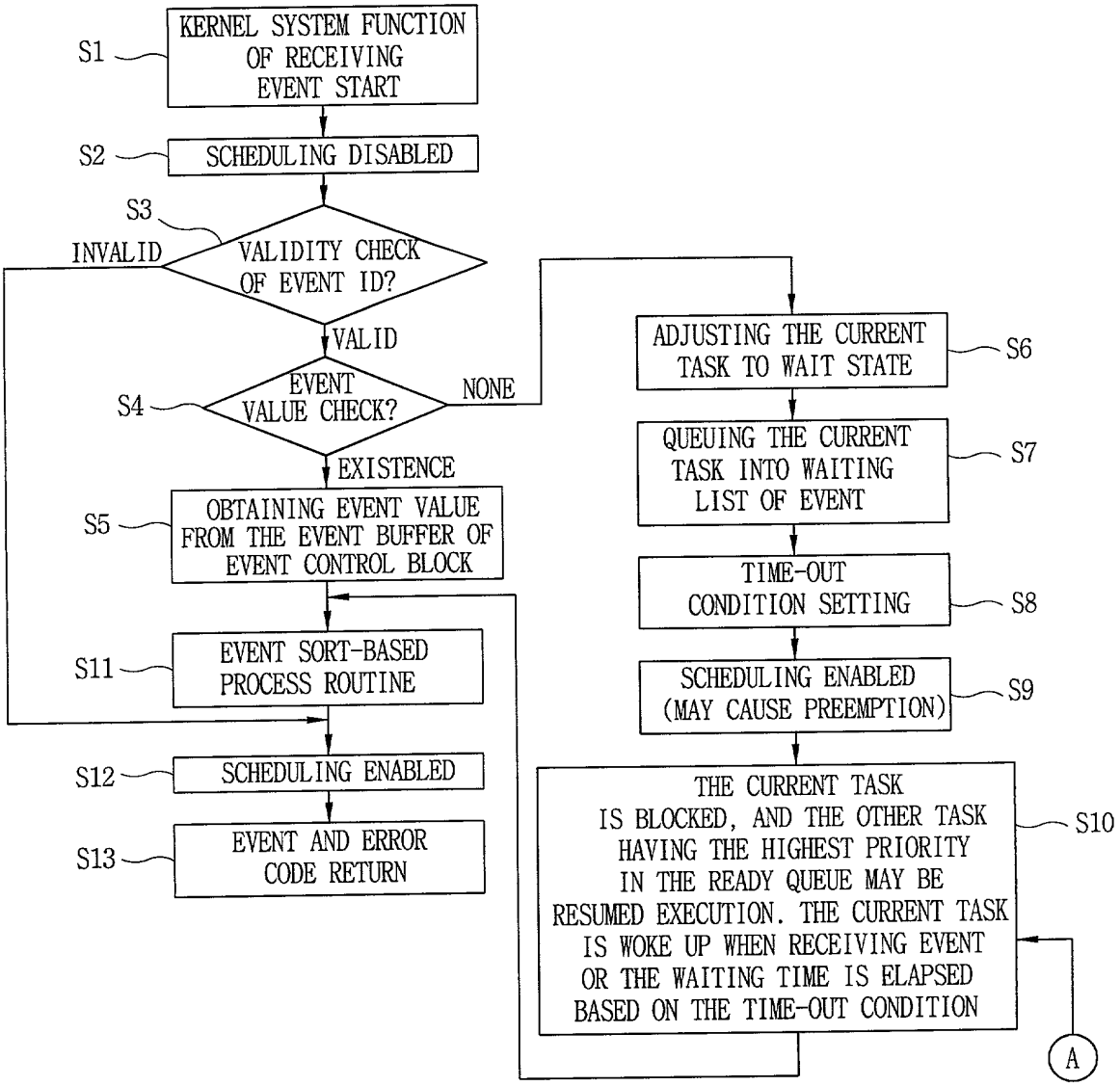

S1
KERNEL SYSTEM FUNCTION OF RECEIVING EVENT START
S2
SCHEDULING DISABLED
S3
INVALID
VALIDITY CHECK OF EVENT ID?
VALID
S4
EVENT VALUE CHECK?
NONE
EXISTENCE
S5
OBTAINING EVENT VALUE FROM THE EVENT BUFFER OF EVENT CONTROL BLOCK
S11
EVENT SORT-BASED PROCESS ROUTINE
S12
SCHEDULING ENABLED
S13
EVENT AND ERROR CODE RETURN
S6
ADJUSTING THE CURRENT TASK TO WAIT STATE
S7
QUEUING THE CURRENT TASK INTO WAITING LIST OF EVENT
S8
TIME-OUT CONDITION SETTING
S9
SCHEDULING ENABLED (MAY CAUSE PREEMPTION)
S10
THE CURRENT TASK IS BLOCKED, AND THE OTHER TASK HAVING THE HIGHEST PRIORITY IN THE READY QUEUE MAY BE RESUMED EXECUTION. THE CURRENT TASK IS WOKE UP WHEN RECEIVING EVENT OR THE WAITING TIME IS ELAPSED BASED ON THE TIME-OUT CONDITION
A

FIG. 3B

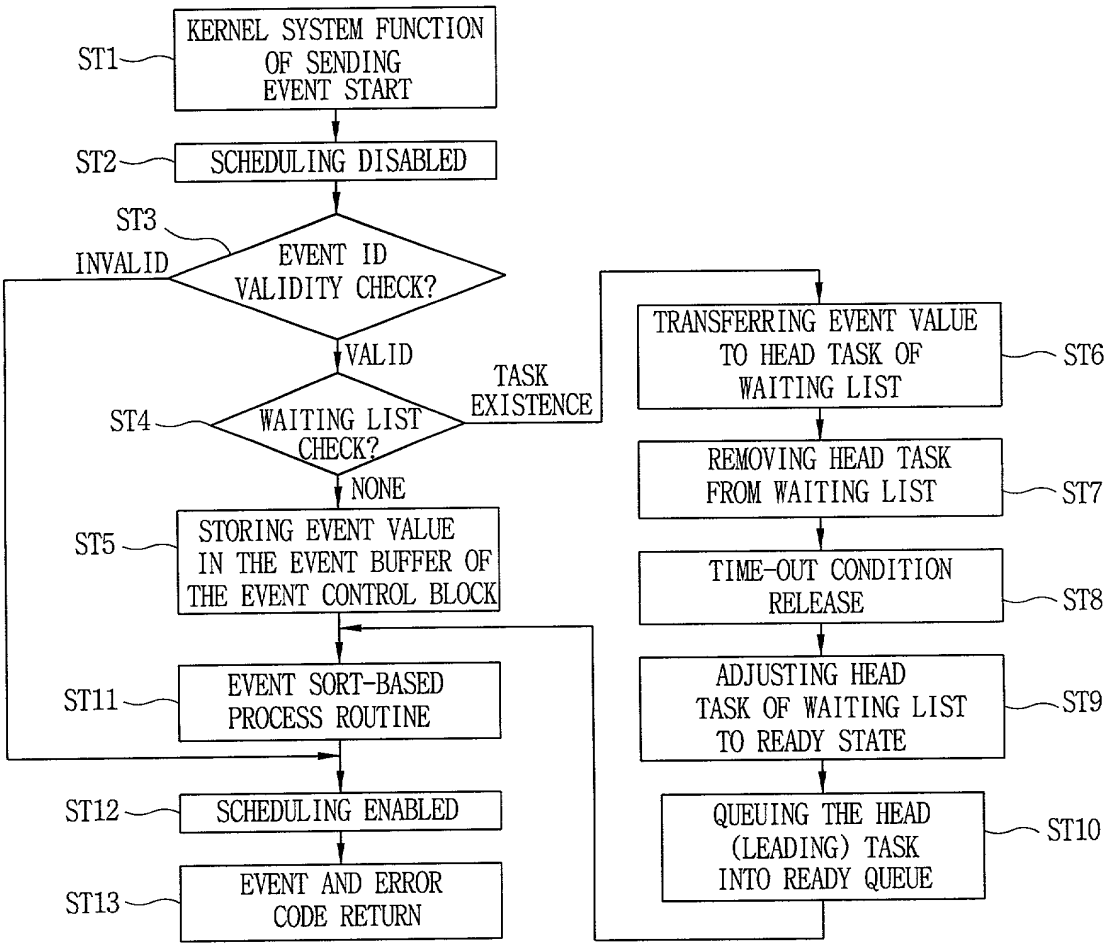
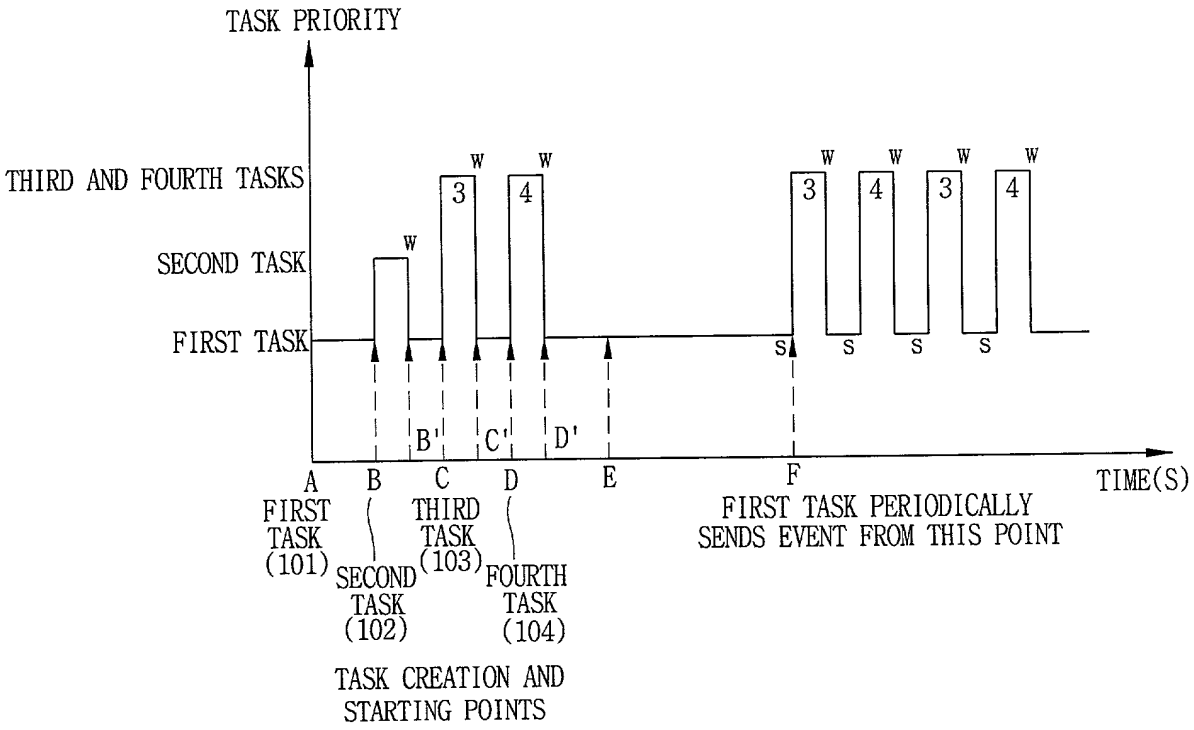

ST1
KERNEL SYSTEM FUNCTION OF SENDING EVENT START
ST2
SCHEDULING DISABLED
ST3
EVENT ID VALIDITY CHECK?
INVALID
VALID
ST4
WAITING LIST CHECK?
TASK EXISTENCE
NONE
ST5
STORING EVENT VALUE IN THE EVENT BUFFER OF THE EVENT CONTROL BLOCK
ST11
EVENT SORT-BASED PROCESS ROUTINE
ST12
SCHEDULING ENABLED
ST13
EVENT AND ERROR CODE RETURN
ST6
TRANSFERRING EVENT VALUE TO HEAD TASK OF WAITING LIST
ST7
REMOVING HEAD TASK FROM WAITING LIST
ST8
TIME-OUT CONDITION RELEASE
ST9
ADJUSTING HEAD TASK OF WAITING LIST TO READY STATE
ST10
QUEUING THE HEAD (LEADING) TASK INTO READY QUEUE

FIG. 4


TASK PRIORITY
THIRD AND FOURTH TASKS
SECOND TASK
FIRST TASK
W
3
4
S
B'
C'
D'
A
B
C
D
E
F
TIME(S)
FIRST TASK (101)
SECOND TASK (102)
THIRD TASK (103)
FOURTH TASK (104)
FIRST TASK PERIODICALLY SENDS EVENT FROM THIS POINT
TASK CREATION AND STARTING POINTS

BIRCH, STEWART, KOLASCH & BIRCH, LLP

PLEASE NOTE: YOU MUST COMPLETE THE FOLLOWING:

COMBINED DECLARATION AND POWER OF ATTORNEY FOR PATENT AND DESIGN APPLICATIONS

ATTORNEY DOCKET NO. 0630-1060P

As a below named inventor, I hereby declare that: my residence, post office address and citizenship are as stated next to my name; that I verily believe that I am the original, first and sole inventor (if only one inventor is named below) or an original, first and joint inventor (if plural inventors are named below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

Insert Title: METHOD FOR IMPLEMENTING EVENT TRANSFER SYSTEM OF REAL TIME OPERATING SYSTEM

Fill in Appropriate Information - For Use Without Specification Attached:

the specification of which is attached hereto. If not attached hereto,

the specification was filed on _____ as
United States Application Number _____; and /or

the specification was filed on _____ as PCT
International Application Number _____; and was
amended under PCT Article 19 on _____ (if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, §1.56.

I do not know and do not believe the same was ever known or used in the United States of America before my or our invention thereof, or patented or described in any printed publication in any country before my or our invention thereof or more than one year prior to this application, that the same was not in public use or on sale in the United States of America more than one year prior to this application, that the invention has not been patented or made the subject of an inventor's certificate issued before the date of this application in any country foreign to the United States of America on an application filed by me or my legal representatives or assigns more than twelve months (six months for designs) prior to this application, and that no application for patent or inventor's certificate on this invention has been filed in any country foreign to the United States of America prior to this application by me or my legal representatives or assigns, except as follows.

I hereby claim foreign priority benefits under Title 35, United States Code, §119 (a)-(d) of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Insert Priority Information: (if appropriate)

Prior Foreign Application(s)

(Number)	(Country)	(Month/Day/Year Filed)	Priority Claimed	
9449/1999	Korea	03/19/1999	☒ Yes	☐ No
			☐ Yes	☐ No
			☐ Yes	☐ No
			☐ Yes	☐ No
			☐ Yes	☐ No

I hereby claim the benefit under Title 35, United States Code, §119(e) of any United States provisional application(s) listed below.

Insert Provisional Application(s): (if any)

(Application Number) _____ (Filing Date) _____

(Application Number) _____ (Filing Date) _____

All Foreign Applications, if any, for any Patent or Inventor's Certificate Filed More Than 12 Months (6 Months for Designs) Prior To The Filing Date of This Application:

Insert Requested Information: (if appropriate)

Country	Application No	Date of Filing (Month, Day, Year)

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, §1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application:

Insert Prior U.S. Application(s): (if any)

(Application Number)	(Filing Date)	(Status - patented, pending, abandoned)

I hereby appoint the following attorneys to prosecute this application and/or an international application based on this application and to transact all business in the Patent and Trademark Office connected therewith and in connection with the resulting patent based on instructions received from the entity who first sent the application papers to the attorneys identified below, unless the inventor(s) or assignee provides said attorneys with a written notice to the contrary:

Terrell C. Birch	(Reg. No. 19,382)	Raymond C. Stewart	(Reg. No. 21,066)
Joseph A. Kolasch	(Reg. No. 22,463)	James M. Slattery	(Reg. No. 28,380)
Bernard L. Sweeney	(Reg. No. 24,448)	Michael K. Mutter	(Reg. No. 29,680)
Charles Gorenstein	(Reg. No. 29,271)	Gerald M. Murphy, Jr.	(Reg. No. 28,977)
Leonard R. Svensson	(Reg. No. 30,330)	Terry L. Clark	(Reg. No. 32,644)
Andrew D. Meikle	(Reg. No. 32,868)	Marc S. Weiner	(Reg. No. 32,181)
Joe McKinney Muncy	(Reg. No. 32,334)	Donald J. Daley	(Reg. No. 34,313)
C. Joseph Faraci	(Reg. No. 32,350)		

Send Correspondence to: **BIRCH, STEWART, KOLASCH & BIRCH, LLP**

P.O. Box 747 • Falls Church, Virginia 22040-0747

Telephone: (703) 205-8000 • Facsimile: (703) 205-8050

PLEASE NOTE: YOU MUST COMPLETE THE FOLLOWING:

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of First or Sole Inventor: Insert Name of Inventor; Insert Date This Document is Signed; Insert Residence; Insert Citizenship; Insert Post Office Address

GIVEN NAME	FAMILY NAME	INVENTOR'S SIGNATURE	DATE*
Min-Seok	JANG	Min-Seok Jang	March 15, 2000

Residence (City, State & Country): Seongnam, Korea

CITIZENSHIP: Republic of Korea

POST OFFICE ADDRESS (Complete Street Address including City, State & Country): Jangantown Keonyoung Apt. 116-203, 66, Bundang-Dong, Bundang-Ku, Seongnam, Kyungki-Do, Korea

Full Name of Second Inventor, if any: see above

GIVEN NAME	FAMILY NAME	INVENTOR'S SIGNATURE	DATE*

Residence (City, State & Country):

CITIZENSHIP:

POST OFFICE ADDRESS (Complete Street Address including City, State & Country):

Full Name of Third Inventor, if any see above

GIVEN NAME	FAMILY NAME	INVENTOR'S SIGNATURE	DATE*

Residence (City, State & Country):

CITIZENSHIP:

POST OFFICE ADDRESS (Complete Street Address including City, State & Country):

Full Name of Fourth Inventor, if any see above

GIVEN NAME	FAMILY NAME	INVENTOR'S SIGNATURE	DATE*

Residence (City, State & Country):

CITIZENSHIP:

POST OFFICE ADDRESS (Complete Street Address including City, State & Country):

Full Name of Fifth Inventor, if any see above

GIVEN NAME	FAMILY NAME	INVENTOR'S SIGNATURE	DATE*

Residence (City, State & Country):

CITIZENSHIP:

POST OFFICE ADDRESS (Complete Street Address including City, State & Country):

* DATE OF SIGNATURE